

Décembre 1986

EDITORIAL

De grandes nouveautés vous attendent pour cette nouvelle année dans l'univers de la micro-informatique:

- tout les micros seront sous MSDOS ou ils ne seront plus.
- les IBM se "Mac-intoshiseront" (GEM-like) et les Mac seront ouverts c'est à dire pourront exploiter les logiciels MSDOS.
- tout viendra de TAIWAN, SINGAPOUR, HONG-KONG bref, les claviers seront jaunes...
- ceux qui sortiront du standard de BIG-BLUE seront condamnés à l'ignorance (d'abord on siffle, puis on tire si on n'obtempère pas).

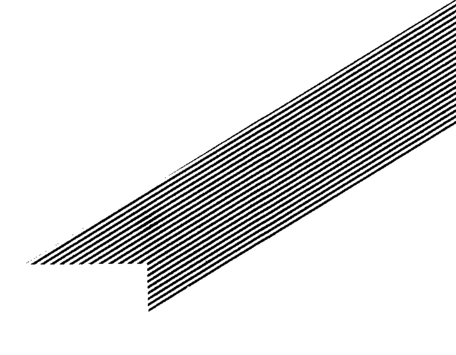
Ceci pour la note d'optimisme coté micro.

Une nouvelle maladie a fait son apparition: le SIDA mental (découvert par le directeur et éditorialiste du FIGARO MAGAZINE): seul remède, attendre que le sujet vieillise ou s'abonne au FIGMAG (à bannir dorénavant de toutes les bonnes salles d'attentes).

Un nouveau langage a fait son apparition: le FORTH 83-Standard (la gestation a été longue); le FIGMAG a passé cette naissance sous silence. Peut-être sommes nous contaminés...

Une nouvelle façon de voir l'univers: le livre de Mr Hubert REEVES, l'heure de s'enivrer. Se dévore sans provoquer d'indigestion. Sa lecture devrait être conseillée à Mr PAUWELLS.

Une nouvelle année enfin, que l'on vous souhaite de tout coeur, pleine de réussite et la concrétisation de tous vos espoirs.



SOMMAIRE

ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS

Tel: (1) 45.42.88.90 (de 10h à 18h)

Une des caractéristiques les plus intéressantes de l'APL, est sa capacité à traiter les variables multidimensionnelles: vecteurs, matrices, tableaux.

Nous n'allons pas faire ici un exposé complet sur le traitement de ces variables, mais simplement tenter d'en donner un aperçu au non-APListe.

Considérons le petit problème suivant: Pierre revient du marché avec 1 pomme, 2 poires et 3 oranges. De son coté, Paul rapporte 3 pommes, 1 poire et pas d'orange. Nous allons créer deux variables:

```
PIERRE ← 1  2  3
PAUL   ← 3  1  0
```

Chacune de ces variables comporte trois valeurs "chaînées" ensemble. On leur donne le nom de "vecteurs" de longueur 3. Si on veut savoir ce qu'il y a dans la variable PIERRE, on frappe:

```
PIERRE
```

et la réponse est: 1 2 3

Si l'on veut savoir la longueur de la variable, on uilise la fonction ρ (rho) sous la forme monadique en frappant:

```
ρ PIERRE
```

La réponse est: 3

PIERRE est un vecteur de longueur 3. Si l'on veut savoir combien de fruits auront été rapportés, il suffira de frapper:

```
PIERRE + PAUL
```

La réponse sera: 4 3 3

C'est à dire 4 pommes, 3 poires et 3 oranges. Si une pomme coûte 1,5 Fr, une poire 1 Fr et une orange 2 Fr (Ndlr: bigre, quelle inflation!!!...), on peut connaître les sommes payées par PIERRE en frappant:

```
PIERRE × 1.5  1  2
```

résultat: 1.5 2 6

PIERRE aura acheté pour 1,5 Fr de pommes, pour 2 francs de poires et pour 6 francs d'oranges.

On aurait pu créer une variable:

```
PRIX ← 1.5  1  2
```

et on aurait obtenu le même résultat en frappant:

```
PIERRE × PRIX
```

On pourra même avoir la somme totale payée par PIERRE en utilisant la fonction "réduction" (/). Sans entrer ici dans des détails de syntaxe, sachons seulement que les symboles +/ placés devant un vecteur provoquent le calcul de la somme des éléments du vecteur:

```
+/ (PIERRE × PRIX)
```

donnera: 9.5

Soit la somme payée par PIERRE. Bien entendu, on peut en faire autant pour PAUL. On aurait pu également obtenir le même résultat en faisant le "produit interne" des vecteurs PIERRE et PRIX, qui s'écrit:

```
PIERRE +.× PRIX
```

Les symboles +.x provoquent les produits des éléments correspondants de chaque vecteur, puis la somme de ces produits. Mais n'allons pas trop loin pour l'instant.

Nous aurions pu rassembler les données relatives à PIERRE et à PAUL en une seule variable bidimensionnelle (tableau) dans laquelle les valeurs relatives à un même individu seraient sur une même ligne, et les valeurs relatives à un même fruit sur une même colonne.

Pour cela, il faut créer une variable à deux lignes et trois colonnes en utilisant la fonction ρ (rho) diadique en frappant:

```
TABLEAU ← 2 3 ρ PIERRE,PAUL
```

La variable TABLEAU contient les valeurs suivantes:

```
1  2  3
3  1  0
```

La fonction ρ monadique permet de contrôler les dimensions de la variable:

```
ρ TABLEAU
```

résultat: 2 3

soit 2 lignes et 3 colonnes.

Le nombre total de fruits rapportés par l'un et l'autre se calcule à l'aide d'une réduction le long des lignes:

```
+/ TABLEAU
```

donne: 6 4

Soit 6 fruits pour PIERRE et 4 pour PAUL. Le nombre de chaque type de fruit est donné par une réduction le long des colonnes:

```
+⌿ TABLEAU
```

réultat: 4 3 3

soit 4 pommes, 3 poires et 3 oranges.

Un produit interne permet de connaître la somme globale payée par chacun:

```
TABLEAU +.× PRIX
```

donne: 9.5 5.5

soit 9,50 Fr pour PIERRE et 5,50 Fr pour PAUL. Une réduction de ce dernier résultat donne la somme totale payée par PIERRE et PAUL.

```
+/ TABLEAU +.× PRIX
```

donne: 15

suite page 8

LOGO

A>type logofich.txt
L'application LOGO que vous trouverez ci-jointe est un répertoire téléphonique grand format, 1000 noms possibles, pouvant servir de support d'étude des fichiers LOGO.

Configuation TO7 + extension 16K + disquette simple face et densité (NdlR: ou TO7-70 + disquette SFSD).

LES GRANDES LIGNES
DE CETTE APPLICATION

Le fichier global des noms et numéros "TEL est rangé sur disquette en 11 tranches gérées par la liste "ARCH.

Seule sera présente en RAM une tranche de 100 noms (à l'aise).

L'échange se fait au moment où ayant tapé le nom (et validé), on cherchera des yeux le numéro de téléphone à inscrire.

Le système présente le ou les numéros des homonymes éventuels pour éviter les redondances.

Les réponses sont classées en liste dans l'ordre croissant des noms au fur et à mesure des entrées et les homonymes, le dernier en tete.

Des corrections par effacement sont possibles.

L'impression du répertoire par tranches.

La sauvegarde de la dernière correction où l'ajout se fait par "OFF avant extinction des feux.

QUELQUES OBSERVATIONS
SUR LES ENCOMBREMENTS MEMOIRE

Un éditeur "OCCUP permet de définir des procédures de test d'occupation qui apportent des renseignements intéressants.

La liste "ELM permet de définir la taille du fichier. Telle que définie, elle génèrera des listes de listes de 10 caractères.

Le premier de chaque mot est un identificateur de ligne, les caractères qui suivent: la colonne:

? DONNE "TST MULTI [ ] 65 116

génère un encombrement

"ELM" réduit à un mot de 9 caractères:
10car x 50 = 500 demandent: 530 octets (il n'y a pas d'objections).

"ELM réduit à deux mots:
2 x 10 x 50 = 1000 demandent: 1340 octets
(on est passé en deux dimensions?).

"ELM à 5 mots:
5 x 10 x 50 = 2500 demandent: 2830 octets
(50 lignes, 5 colonnes, rien à dire).

LOGO ouvre une autre possibilité, celle de l'indirection que nous avons testée.

? DONNE "TST SIMPLE [ ] 65 116

génère une liste de mots définis par le premier de "ELM auquel on "DONNE le restant de cette liste. Le meme nombre de caractères n'est généré que dans le dernier cas, mais l'organisation et l'exploitationne sont plus du tout le memes , l'encombrement non plus.

Pour 10 x 50 + 4 x 10 x 50 = 2500 caractères, l'encombrement passe à 3080 octets.

LES ENCOMBREMENTS
LORS DE TRAITEMENT

Ils sont particulièrement intéressants à connaitre. Deux méthodes sont possibles, à ma connaissance, et seulement deux:

- la récursivité non terminale permet de supprimer ou modifier un élément quelconque

```
EC "VRP-19

POUR INIT
DONNE "TEL [[AAAAA A] [ZZZZZ Z]]
FIN

POUR VERSE :ME
SI VIDE? :ME [STOP]
DONNE "TEL ORDRE INSR :TEL PH PREM PREM :ME PREM SP PREM :ME
VERSE SP :ME
FIN

POUR INSR :M :N
SI PM<SM PREM :N PREM PREM :M [RENDS MP :N :M]
RENDS INSR +LIF :M :N
FIN

EC "VRP-20

POUR HOMON :M :NOM
SI EGAL? :NOM PREM PREM :M [VISUL :AFI PREM :M :LON TAPE "OUI? SI EGAL? "
O LISCAR [EC [] RENDS :M] [EC []]]
SI PM<SM :NOM PREM PREM  :M [TAPE [tel:] RENDS MP PH :NOM PREM LL :M]
RENDS HOMON +LIF :M :NOM
FIN

POUR ORDRE :M
SI ASCII PREM PREM :M < ASCII PREM DER :M [RENDS :M]
RENDS ORDRE +LIF :M
FIN

POUR PM<SM :P :S
SI VIDE? :P [RENDS VRAI]
SI VIDE? :S [RENDS FAUX]
SI EGAL? PREM :P PREM :S [RENDS PM<SM  SP :P SP :S]
RENDS PLP? ASCII PREM :P ASCII PREM :S
FIN

POUR +LIF :L
RENDS MD PREM :L SP :L
FIN

POUR -LIF :L
RENDS MP DER :L SD :L
FIN

EC "VRP-21
DONNE "AFI [nom: tel:]

DONNE "LON [16 11]

POUR E
TAPE [Premier caractere ?]
DONNE "NOM PREM LL
SI NON NOM? "DSK [NARCH]
SI NON TAR? :DSK ASCII :NOM [OFF NARCH]
ENUM 1 SP :TEL
TAPE [NUM A EFFACER ?]
DONNE "NUM PREM LL
SI NOMBRE? :NUM [DONNE "TEL ORDRE ENLI :NUM + 1 :TEL]
FIN

POUR ENLI :N :L
SI EGAL? ITEM :N :TEL PREM :L [RENDS SP :L]
RENDS ENLI :N +LIF :L
FIN

POUR VISUL :A :M :L
SI VIDE? :A [STOP]
TAPE MOT PREM :A PREM :M
REPETE PREM :L - CO PREM :M [TAPE CAR 9]
VISUL SP :A SP :M SP :L
FIN

POUR CO :M
SI VIDE? :M [RENDS 0]
RENDS 1 + CO SP :M
FIN
```

DONNE "TST DEBAL :TST

provoque l'affichage de la place disponible après génération de "TST sans différentes formes. Il faut noter que l'occupation est considérable et indépendante de la forme, seule la longueur de liste importe (le nombre de lignes).

- la récursivité terminale avec une gestion circulaire permet le meme travail, avec une occupation infime par rapport à la première méthode.

DONNE "TST CIRCUM : TST COMPTE :TST

4500 octets dans la 1ère méthode
430 octets dans la dernière méthode.

La première méthode permet une définition très claire, très élégante, mais à réserver pour les listes courtes.

LES PROCEDURES PRINCIPALES

Ajoute: POUR A
permet d'ajouter un nouveau nom avec gestion circulaire de la liste "TEL. CElle-ci a été ouverte avec des mots qui seront des identificateurs de début et de fin. Quelle que soit la déformation de cette liste, la procédure ORDRE rendra une liste avec ces identificateurs correctement rétablis.

Menu: POUR M
la procédure M est une commande vecteur qui écrit le menu en fonction du contenu de la liste "MENU, ceci permettant de valider un choix et un seul. Elle pourrait être éventuellement complétée par des effacements de procédures indésirables le cas échéant.

Nota: la limite en longueur de ce fichier n'est pas tellement l'occupation mémoire, mais le temps mis par le système pour installer une nouvelle tranche d'archive "TEL en mémoire. Ce temps n'est pas négligeable pour les grands fichiers.

```
POUR ENUM :P :L
SI VIDE? SD :L [STOP]
TAPE :P VISUL :AFI PREM :L :LON EC "
ENUM 1 + :P SP :L
FIN

EC "VRP-22
DONNE "ARCH [[64 66 A-A/TEL] [65 67 B-B/TEL] [66 68 C-C/TEL] [67 69 D-D/T
EL] [68 71 E-F/TEL] [70 72 G-G/TEL] [71 77 H-L/TEL] [76 78 M-M/TEL] [77 8
1 N-P/TEL] [80 84 Q-S/TEL] [83 91 T-Z/TEL]]

POUR A
TAPE "nom?:
DONNE "NOM PREM LL
SI NON NOM? "DSK [COLOR NARCH SUITE STOP]
SI TAR? :DSK ASCII :NOM [SUITE] [OFF NARCH SUITE]
FIN

POUR SUITE
DONNE "TEL ORDRE HOMON :TEL :NOM
FIN

POUR NARCH
DONNE "DSK SLCT :ARCH ASCII :NOM
RAMENE DER :DSK
FIN

POUR SLCT :LA :A
SI VIDE? :LA [RENDS []]
SI TAR? PREM :LA :A [RENDS PREM :LA] [RENDS SLCT SP :LA :A]
FIN

POUR TAR? :L :A
RENDS ET PREM :L < :A PREM SP :L > :A
FIN

POUR OFF
DETRUIS DER :DSK
SAUVE DER :DSK ["MEMO]
FIN

POUR COLOR
TAPE MOT CAR 27 CAR 97
TAPE MOT CAR 27 MOT CAR 32 CAR 64
TAPE MOT CAR 27 MOT CAR 32 CAR 83
FIN

EC "VRP-23

POUR I
EC "IMPRESSION
SI NOM? "TEL [EFN "TEL]
SELAR :ARCH
SORTIE 2
EC " ELEM SP :TEL
SORTIE 1
FIN

POUR SELAR :L
SI VIDE? :L [STOP]
TAPE PH "de PH CAR 1 + PREM PREM :L PH "a PH CAR PREM SP PREM :L - 1 [OUI
 ?]
SI EGAL? "O LISCAR [RAMENE DER PREM :L STOP] [EC "]
SELAR SP :L
FIN

POUR ELEM :L
SI VIDE? SD :L [STOP]
VISUL :AFI PREM :L :LON EC "
ELEM SP :L
FIN

POUR M
COLOR
EXEC :MENU
FIN

POUR !>TEL
DONNE "MENU [EC ['A'joute] EC ['E'dite,efface] EC ['OFF'sauve disque]]
SAUVE ">TEL [HOMON ORDRE PM<SM +LIF  "AFI "LON E ENLI VISUL CO ENUM A SU
ITE NARCH SLCT TAR? OFF COLOR "MENU "ARCH  M ]
FIN

POUR !<TEL
DONNE "MENU [EC ['I'mprime]]
SAUVE "<TEL ["AFI "LON VISUL CO  COLOR "MENU "ARCH  M I SELAR ELEM]
FIN
```

```
EC "OCCUP
DONNE "ELM [aaaaaaaaa bbbbbbbbb ccccccccc ddddddddd eeeeeeeee]

POUR ALIGNE :L :CD :ID
SI VIDE? :ID [RENDS :L]
RENDS MP MOT CAR :CD PREM :ID ALIGNE :L :CD SP :ID :L
FIN

POUR LIDOLI :P :CD :ID
DONNE "P MOT CAR :CD PREM :ID
DONNE :P ALIGNE [] :CD SP :ID
RENDS :P
FIN

POUR SIMPLE :L :CD :CF
SI EGAL? :CD :CF [RENDS :L]
RENDS MP LIDOLI " :CD :ELM SIMPLE :L 1 + :CD :CF :L
FIN

POUR MULTI :L :CD :CF
SI EGAL? :CD :CF [RENDS :L]
RENDS MP ALIGNE [] :CD :ELM MULTI :L 1 + :CD :CF :L
FIN

POUR DEBAL :L
SI VIDE? :L [EC PH "DISPONIBLE: PLACE RENDS []]
RENDS MP PREM :L DEBAL SP :L
FIN

POUR +LIF :L
RENDS MD PREM :L SP :L
FIN

POUR CIRCUM :L :N
SI EGAL? 0 :N [RENDS :L]
RENDS +LIF :L 1 - :N
FIN
```

EDITEUR PLEIN ECRAN

par Alain BARRAUD

FIG-Forth pour AMSTRAD

```
SCR # 36
;S          Pour les mordus du Forth-Amstrad:

voici  un  Editeur  Pleine-Page  qui  utilise les  commandes de
l'editeur du Fig.  Il  est  bien  plus  rapide et confortable a
utiliser que celui paru dans Jedi 25  (pardon a R.Jeannin, mais
merci pour sa copie d'ecran avec l'utilitaire de calculs en LM).
N'utilisez que les commandes citees apres vous etre assures que
les codes  Ascii de votre clavier concordent avec ceux utilises
ici ( sinon servez-vous du SETUP.COM ou modifiez-les ).
En cas de bugs residuels ou pour toute autre information, voici
ou me joindre:
                BARRAUD Alain       (membre Jedi)
                12 rue des sapins
                77210 AVON

      tel: (16.1) 60.72.25.43  (19H-22H ou Week-end)
```

```
SCR # 37
( Commandes de l'Editeur Pleine Page                 ABSEP86 )
                                                            ;S
FLECHES-CURSEUR           deplacement dans l'ecran vers
                                    - haut
                                    - bas
                                    - gauche
                                    - droite

<CTRL> FLECHE-HAUT       amene le curseur en debut de ligne  0
<CTRL> FLECHE-BAS        amene le curseur en debut de ligne 15
<CTRL> FLECHE-GAUCHE     amene le curseur en debut de ligne
<CTRL> FLECHE-DROITE     amene le curseur en fin   de ligne

<SHIFT> FLECHE-HAUT      liste l'ecran n-1
<SHIFT> FLECHE-BAS       liste l'ecran n+1
```

```
SCR # 38
( Commandes de l'Editeur Pleine Page                 ABSEP86 )
                                                            ;S
<CTRL> X         remplit l'ecran avec le caractere 'espace' .
<CTRL> E         efface la ligne ou se trouve le curseur .
<CTRL> D         ote la ligne ou se trouve le curseur ,
                 celles du dessous remontent .
<CTRL> S         insere une ligne vierge , celles du dessous
                 descendent, la 15eme est perdue .
<CTRL> H         memorise dans PAD la ligne courante .
<CTRL> R         recopie en ligne courante la ligne memorisee
                 par <ctrl> H ( si elle est toujours dans PAD ).
<CTRL> I         insere en ligne courante la ligne memorisee
                 dans PAD, la 15eme est perdue ( meme remarque ).
<CTRL> <TAB>     tabulation de 8 .NE PAS UTILISER <TAB> SEUL !!
                 a cause de son code ASCII identique a <CTRL> I .
<CTRL> Q         retour en Forth, a force d'y penser.
```

```
SCR # 39
( Commandes de l'Editeur Pleine Page                 ABSEP86 )
                                                            ;S
<DEL>                 efface le caratere sous le curseur, les
                      suivants se decalent vers la gauche, un
                      espace est ajoute en fin de ligne.

<COPY>                insere un espace a l'endoit du curseur,
                      le dernier caractere de la ligne est perdu.

<ENTER>               fait passer en debut de ligne suivante.

A part ces commandes, le texte s'inscrit la ou se trouve le
curseur.
Ne pas oublier de faire FLUSH apres avoir quitte l edition.
```

```
SCR # 40
( Editeur pleine page                                  ABSEP86 )

EDITOR DEFINITIONS   DECIMAL

: BEEP               ( --- ! fait 'bip' )
   7 EMIT ;

: GAUCHE              ( --- ! decremente R# de 1 )
   R# @ 0= IF   BEEP
          ELSE -1 R# +!
          THEN ;
: DROITE              ( --- ! incremente R# de 1 )
   R# @ 1023 = IF   BEEP
             ELSE 1 R# +!
             THEN ;
                                         -->
```

```
SCR # 41
( Editeur pleine page suite                            ABSEP86 )
: HAUT          ( --- ! decremente R# de 64 )
   R# @ 64 < IF     BEEP
         ELSE   -64 R# +!
         THEN    ;
: BAS            ( --- ! incremente R# de 64 )
   R# @ 959 > IF     BEEP
         ELSE   64 R# +!
         THEN    ;
: DEB-L15          ( --- ! R# indique le debut de ligne 15 )
   960   R#   ! ;
: DEB-L            ( --- ! R# indique le debut de ligne )
   #LEAD  MINUS  R#  +!  DROP  ;
: FIN-L            ( --- ! R# indique la fin de ligne )
   #LAG   1  -  R#  +!   DROP  ;
                                     -->
```

```
SCR # 42
( Editeur pleine page suite                            ABSEP86 )
: <ENTER>        ( --- ! R# indique le debut de ligne dessous )
   R# @ 959 > IF     DEB-L15  BEEP
            ELSE   #LAG     R#   +!   DROP
            THEN      ;

: CURSEUR        ( --- ! affiche le curseur )
   #LOCATE   1+  SWAP 1+ LOCATE ;

: LIGNE             ( --- ligne )
   #LOCATE  SWAP  DROP ;

: .LIGNE           ( ligne --- ! affiche la ligne a sa place )
    DUP   1+ 1     LOCATE
    SCR  @   .LINE     ;
                                    -->
```

```
SCR # 43
( Editeur pleine page suite                            ABSEP86 )
: .ECRAN    ( --- ! liste l'ecran )
   16 0 DO      FORTH I .LIGNE
        LOOP      ;

: ERA-LIGNE ( ligne --- ! efface la ligne courante de l'ecran )
   1+  1 LOCATE 18 EMIT ;

: CTRL_E     ( --- ! efface la ligne en buffer et ecran )
  LIGNE DUP E ERA-LIGNE ;

: CTRL_D     ( --- ! ote la ligne courante buffer et ecran )
  LIGNE DUP D 1+ 1 LOCATE 20 EMIT .ECRAN ;
```

SCR # 44

```
( Editeur pleine page suite                              ABSEP86 )

: CTRL_X  ( --- | efface l'ecran buffer et ecran )
    SCR @ CLEAR CLS ;

: EN-TETE          0 #WINDOW 1 20 LOCATE
  ." Editeur Pleine Page : SCR # " SCR @ 3 .R   ;

: SCR-1              ( --- | montre l'ecran precedent )
   SCR @  1 = IF   BEEP
               ELSE -1 SCR +! EN-TETE 1 #WINDOW CLS TOP .ECRAN
               THEN ;
: SCR+1              ( --- | montre l'ecran suivant )
   SCR @ 179 = IF   BEEP
               ELSE  1 SCR +! EN-TETE 1 #WINDOW CLS TOP .ECRAN
               THEN ;                -->
```

SCR # 45

```
( Editeur pleine page suite                              ABSEP86 )

: INS                ( --- | insere un espace sous le curseur )
   #LAG PAD SWAP CMOVE
   BL #LAG DROP C!
   PAD    #LAG 1 - >R 1+ R> CMOVE
   CURSEUR 18 EMIT #LAG TYPE UPDATE ;


: DEL                ( --- | ote le caractere sous le curseur )
   PAD C/L 1+ BL FILL
   #LAG PAD SWAP CMOVE
   PAD 1+ #LAG CMOVE
    CURSEUR 18 EMIT #LAG TYPE UPDATE ;

-->
```

SCR # 46

```
( Editeur pleine page suite                              ABSEP86 )
: CTRL_TAB R# @ 1016 < IF   8 R# +!         ( --- | tabule de 8 )
                  ELSE BEEP THEN ;

: ECRIT?            ( c --- | ecrit un caratere imprimable )
  DUP 31 < SWAP DUP 127 > ROT OR
    IF    DROP
    ELSE  DUP #LEAD + C! EMIT
    THEN  DROITE  UPDATE  ;

: CTRL_H    ( memorise la ligne courante dans PAD )
   LIGNE H ;

: CTRL_R    ( recopie en ligne courante la ligne memorisee )
   LIGNE  DUP DUP
   ERA-LIGNE EDITOR R .LIGNE ;             -->
```

SCR # 47

```
( Editeur pleine page suite                              ABSEP86 )

: CTRL_I ( insere la ligne memorisee. la 15eme est perdue )
   LIGNE EDITOR I
   LIGNE 1+ 1 LOCATE 20 EMIT .ECRAN ;



: STOP  ( --- | quitte l'edition )
    2 MODE
   [COMPILE] FORTH [COMPILE] DEFINITIONS  QUIT ;


-->
```

SCR # 48

```
( Editeur pleine page suite                              ABSEP86 )

: INIT          ( theatre des prochaines scenes )
   2 MODE
   0 #WINDOW    25 3 80 5 WINDOW
   EN-TETE
   16 0 DO
         FORTH I 4 + 1   LOCATE FORTH I 3 .R
         FORTH I 4 + 70  LOCATE FORTH I  .
       LOOP
   1 #WINDOW    23 6 73 9  WINDOW
   CLS
   TOP
   .ECRAN  CURSEUR  ;
-->
```

SCR # 49

```
( Editeur pleine page suite                              ABSEP86 )
: CHOIX
   BEGIN  KEY
 CASE
243 ( fleche droite )       OF DROITE    ENDOF
242 ( fleche gauche )       OF GAUCHE    ENDOF
240 ( fleche haut )         OF HAUT      ENDOF
241 ( fleche bas )          OF BAS       ENDOF
248 ( ctrl fleche haut )    OF TOP       ENDOF
249 ( ctrl fleche bas )     OF DEB-L15   ENDOF
250 ( ctrl fleche gauche )  OF DEB-L     ENDOF
251 ( ctrl fleche droite )  OF FIN-L     ENDOF
245 ( shift fleche bas )    OF SCR+1     ENDOF
244 ( shift fleche haut )   OF SCR-1     ENDOF
225 ( ctrl tab )            OF CTRL_TAB  ENDOF
               -->
```

SCR # 50

```
( Editeur pleine page suite                              ABSEP86 )
 24 ( ctrl X )              OF CTRL_X    ENDOF
  5 ( ctrl E )              OF CTRL_E    ENDOF
  4 ( ctrl D )              OF CTRL_D    ENDOF
 19 ( ctrl S )              OF CTRL_S    ENDOF
  8 ( ctrl H )              OF CTRL_H    ENDOF
 18 ( ctrl R )              OF CTRL_R    ENDOF
  9 ( ctrl I )              OF CTRL_I    ENDOF
 17 ( ctrl Q )              OF STOP      ENDOF
224 ( COPY )                OF INS       ENDOF
127 ( DEL )                 OF DEL       ENDOF
 13 ( ENTER )               OF <ENTER>   ENDOF
  DUP  ECRIT?
          ENDCASE    CURSEUR
  AGAIN ;
               -->
```

SCR # 51

```
( Editeur pleine page fin                                ABSEP86 )
: EDIT
       DECIMAL SCR ! INIT CHOIX ;

FORTH DEFINITIONS DECIMAL

: ED EDITOR EDIT  ;              FORTH

;S

      +---------------------------------------------------+
      | syntaxe :      n  ED        ( n est l'ecran a editer) |
      +---------------------------------------------------+
```

FONCTION HORLOGE pour F83 sous MSDOS

par A.JACCOMARD

1

```
\ Chargement.


  1 5 +THRU

  55 VDO
CR .( Horloge chargée.)
CR .( Pour changer date/heure, utilisez DATE-HEURE.)
CR .( Pour avoir date/heure, utilisez NOW.)
  48 VDO

  6 VIEWS HORL.BLK  \ place ce fichier dans VIEW-FILES.
```

9

```
\ HORLOGE.                                                   10Oct86JaD

   Ce programme suppose la présence sur une carte multifonctions
de l'horloge "temps réel" du PC-XT, accessible par les fonctions
42 à 45 de l'interruption 21H.
   fonct. 42 : lecture de la date, CX = année, DX = mois/jour.
     "     43 : positionne la date, mêmes valeurs.
     "     44 : lecture de l'heure, CX = hh/mm, DX = ss/100.
     "     45 : positionne l'heure, mêmes valeurs.
```

2

```
\ Mois et Jours.                                 10Oct86JaD

: "ARRAY   (S lgr -- : compil. ; -- adr lgr : exécut. )
  CREATE  C,  ASCII " WORD  COUNT >R HERE R@ MOVE R> ALLOT
  DOES>   COUNT >R SWAP R@ * + R> ;

3 "ARRAY "MOIS "JanFevMarAvrMaiJunJuiAouSepOctNovDec"
3 "ARRAY "DAY "DimLunMarMerJeuVenSam"

CODE LIS-H  (S fonct --- heure ou date empilée )
  AX POP  AL AH MOV    33 INT
  DX PUSH     \ sec. et 1/100 ou mois/jour
  CX PUSH     \ heure et min. ou année.
  AL PUSH     \ n° du jour de la semaine
  NEXT   C;
```

10

```
\ Mois et Jours.                                             22Oct86JaD

"ARRAY  mot de définition pour les tableaux de chaînes.

"MOIS   (S n --- )  n de 0 à 11
        tableau des noms de mois.

"DAY    (S n --- )  n de 0 à 6
        tableau des noms de jour.

LIS-H   lecture de la date (fonct=42) ou de l'heure (fonct=44).
        Retourne les valeurs sur la pile.
```

3

```
\ Horloge: CLK!, CLK@.                           22Oct86JaD

  VARIABLE CLK!   CLK! 7 ALLOT
: CLK@
  42 LIS-H                           \ lecture de la date
        256 /MOD DROP  CLK!    C!    \ Sj
  SWAP  256 /MOD SWAP  CLK! 1+ C!    \ JJ
                       CLK! 2+ C!    \ MM
                       CLK! 3 + !    \ AA
  44 LIS-H                           \ lecture de l'heure
   DROP  CLK! 5 + 4 BOUNDS     \ calcule limites de boucle
  DO  256 /MOD  I C!  I 1+ C!  2 +LOOP ; \ place HHmmss en CLK!

: H?  \ teste la présence de l'horloge.
  CLK@ CLK! 1+ C@ 0= IF CR ." Horloge inactive."
     FALSE ELSE TRUE THEN ;
```

11

```
\ Horloge: CLK!, CLK@.                                       22Oct86JaD

CLK!    table contenant date/heure.

CLK@    lit la date et l'heure de l'horloge et les place
        dans CLK!:
        le 1er octet contient le n° du jour de la semaine
           2, 3, 4 et 5èmes pour le jour, le mois et l'année;
           6, 7, 8 et 9èmes pour heure, minute, sec. et 1/100.

H?     teste la présence de l'horloge "temps réel", en supposant
       qu'en son absence le contenu de l'octet est nul.
```

4

```
\ Horloge: affichage.

: CLK#    (S n --- )
  CLK! + C@ S>D # #  ;
: (DATE)   (S --- adr lgr )
  CLK@  <# 3 CLK! + @ 100 /MOD DROP S>D #S
  2 CLK! + C@ 1- "MOIS DUP NEGATE HLD +! HLD @ SWAP CMOVE
  1 CLK# #> ;
: (HEURE) (S --- adr lgr )
  CLK@  <# 7 CLK# ASCII : HOLD 6 CLK# ASCII : HOLD
  5 CLK# #> ;
: JOUR   CLK@ CLK! C@ "DAY TYPE SPACE ;
: DATE   (DATE) TYPE SPACE ;
: HEURE  (HEURE) TYPE SPACE ;
: NOW   H? IF 115 CURS  55 VDO  51 24 AT ." le "
  JOUR DATE  ." à " HEURE  48 VDO  117 CURS  THEN  ;
```

12

```
\ Horloge: affichage.                                        22Oct86JaD

CLK#   convertit le n ième oct. de CLK! en ses codes ASCII.

(DATE) construit une chaîne contenant la date, format  JJMmmAA

(HEURE) construit une chaîne contenant l'heure, format  HHmmss

JOUR    affiche le jour de la semaine ( Lun, Mar, ... )
DATE       "    la date
HEURE      "    l'heure

NOW        "    date/heure en bas de l'écran.
```

5

```
0 \ EDITOR ID automatique.                          10Oct86JaD

: (QUI)   (S -- )   " JaD" ;

: QUI   (S -- )   (QUI) TYPE SPACE ;

: SET-ID   (S -- )
   H?
   IF  (DATE) [ EDITOR ] ID SWAP CMOVE  (QUI) ID 7 + SWAP CMOVE
   THEN   HELLO  ;

' SET-ID IS BOOT
```

13

\ EDITOR ID automatique. 22Oct86JaD

(QUI) empile adr et lgr de la chaîne contenant l'identificateur de l'utilisateur.

QUI l'imprime.

SET-ID remplace la routine usuelle de "boutage". Après exécution de HELLO, et si l'horloge "temps réel" est présente, le tampon de l'éditeur ID contient la date et les initiales de l'utilisateur, qui seront placées par STAMP dans chaque écran modifié.

La phrase " ' SET-ID IS BOOT " permet, après sauvegarde du système, une mise à jour automatique du tampon d'éditeur.

6

```
0 \ Mise à date/heure.                              22Oct86JaD

CODE   MIS-D   (S JJ MM AAaa --- fl )
   CX POP  AX POP AL DH MOV  AX POP  AL DL MOV  43 # AH MOV
   33 INT  AH AH SUB  AX PUSH  NEXT  C;

CODE  MIS-H   (S HH mm --- fl )
   AX POP  AL CL MOV  AX POP  AL CH MOV  45 # AH MOV
   DX DX SUB  33 INT  AH AH SUB  AX PUSH  NEXT  C;

: INPUT?   ( -- [n] f )
   QUERY BL WORD NUMBER? NIP DUP 0= IF NIP THEN  ;
-->
```

14

\ Mise à date/heure. 22Oct8222Oct86JaD

MIS-D les valeurs empilées JJ MM AAaa sont placées dans les registres de l'horloge.

MIS-H de même pour l'heure HH mm (les secondes sont mises à 0)

Au retour, la pile contient un drapeau vrai (-1) si les valeurs de date ou d'heure ne sont pas valides.

INPUT? entrée d'un nbr; empile seulement un drapeau faux s'il y a eu entrée d'un autre caractère: la valeur concernée n'est alors pas modifiée.

7

```
0 \ Mise à date/heure.                              22Oct86JaD

: DATE-HEURE   (S -- )
   H? NOT  IF  EXIT  THEN
    CR ." Date/heure courantes : "  DATE  HEURE  CR
    CR ." Jour ? "     INPUT? IF  1 CLK! +  C!  THEN
    CR ." Mois ? "     INPUT? IF  2 CLK! +  C!  THEN
    CR ." Année ? "    INPUT? IF  3 CLK! +  C!  THEN
    CR ." Heure ? "    INPUT? IF  5 CLK! +  C!  THEN
    CR ." Minute ? "   INPUT? IF  6 CLK! +  C!  THEN
CR  ." Frappez une touche pour prise en compte." KEY DROP CR
    CLK! 1+ C@   CLK! 2+ C@  3 CLK! + @     MIS-D
    IF ." Date non valide, recommencez." CR CR  RECURSE  THEN
    CLK! 5 +  C@  CLK! 6 + C@    MIS-H
    IF ." Heure non valide, recommencez." CR CR  RECURSE  THEN
      NOW  ;
```

15

\ Mise à date/heure. 22Oct86JaD

DATE-HEURE demande de la date et de l'heure pour mise à jour.
Le jour JJ est un nbr compris entre 1 et 31.
Le mois MM est un nbr compris entre 1 et 12.
L'année est comprise entre 1980 et ... 2047.
L'heure HH est comprise entre 0 et 23, les minutes mm entre 0 et 59; ne pas entrer de secondes.
Si un caractère autre qu'un chiffre est entré, la valeur courante n'est pas modifiée.
La prise en compte de ces valeurs se fait à l'appui d'une touche.
Leur validité est vérifiée par le BIOS: en cas d'erreur, il faut recommencer.
Vérification à posteriori par NOW .

suite de la page 2

Ces quelques exemples très simples (et peu réalistes...) permettent, sans entrer dans les détails de la syntaxe, de donner une petite idée de la facilité et de la puissance que procure le langage APL dans la manipulation des variables multidimensionnelles. Pour faciliter la compréhension, nous avons utilisé des variables de faibles dimensions, mais les procédures utilisées sont indépendantes des dimensions des variables, lesquelles ne sont pratiquement limitées que par la mémoire disponible.

D'autre part, nous n'avons fait qu'effleurer le sujet, et l'APL possède bien d'autres possibilités de calcul.

Notamment, l'inversion d'une matrice qui est une affaire d'état avec d'autres langages, se programme en APL par un seul signe opératoire en forme de domino.

F.ESPINASSE

Bruce W. Walker, San Pedro, California
Forth Dimensions V/6
Traduction Eric Aubourg

Cet article donne une manière simple d'ajouter des structures de données au Forth, comme en PL/I, en COBOL ou en PASCAL. Il contient trois parties : Premièrement, ce que sont les structures de données et comment on peut les utiliser en Forth, deuxièmement comment fonctionnent les définitions Forth et troisièmement les définitions elles-mêmes.

Les structures de données.

Les programmes Forth utilisent en général uniquement deux types de données : les nombres et les tableaux de nombres. Alors que cela convient pour du calcul scientifique, les programmeurs de gestion savent depuis des temps immémoriaux (c.a.d. depuis 1960) que beaucoup de programmes sont plus faciles à concevoir et à écrire si l'on utilise le concept de structure de données. Une structure de données est une collection hétérogène de données, et on peut la concevoir soit comme une seule entité soit comme une collection avec des parties ayant des noms, selon la nécessité du moment.

Alors que le Forth a peu de possibilités de structures de données implémentées en standard, il a tous les outils nécessaires pour ajouter de nouveaux types de données. D'autres auteurs ont travaillé sur les piles utilisateur[5], les nombres complexes[2], la quadruple précision[4] et les chaînes de caractères[3, 6]. La chose qui se rapproche le plus de structures de données en Forth est l'article sur la conception d'une base de données en Forth[4]; cet article est une mine d'or d'idées et tout programmeur Forth sérieux devrait le lire au moins une fois par an.

```
DECLARE 1 EMPLOYE,
        2 NOM CHARACTER (16),
        2 AGE FIXED BINARY (15,0);
```

Figure 1.

En PL/I, un programme peut avoir une déclaration comme celle de la figure 1. Cela signifie que EMPLOYE est composé de deux parties, NOM et AGE. On peut faire appel aux champs individuellement par EMPLOYE.NOM et EMPLOYE.AGE.

```
DECLARE 1 BOSS,
        2 NOM CHARACTER(16),
        2 AGE FIXED BINARY(15,0);
```

Figure 2.

L'avantage par rapport à deux variables séparées est que EMPLOYE peut être utilisé comme un tout. Par exemple, supposons que nous ayons aussi la déclaration de la figure deux. Alors l'instruction

BOSS = EMPLOYE
assigne les deux champs d'un coup.

```
18 SS.EMPLOYE
  S.MOVE EMPLOYE.MOVE
16 S.FLD NOM
 2 S.FLD AGE
```

Figure 3.

Pour faire la même chose en Forth, nous avons besoin de déclarer la structure, les champs, et créer des routines pour faire les assignements. Cela s'avère être très facile avec CREATE et DOES>. Les définitions sont données dans la section concluant cet article. L'exemple précédent de structure donnerait en Forth la figure 3. (Je n'ai pas inclu les fonctions de transfert, qui sont optionnelles.) Les références aux adresses des champs sont alors:
EMPLOYE NOM
EMPLOYE AGE
Et l'assignation en bloc est effectuée par:
EMPLOYE BOSS EMPLOYE.MOVE
(Notez que l'on ne redéclare pas nom et âge pour boss.)

Deux généralisations des structures de données apparaissent en PL/I. Premièrement, les éléments peuvent être des tableaux, comme dans la figure 4.

```
DECLARE 1 EMPLOYE(100),
        2 NOM CHARACTER(16),
        2 AGE FIXED BINARY(15,0);
```

Figure 4.

Ceci déclare un tableau de structures de données, ici 100 employés. Un programme peut alors contenir les instruction de la figure 5.

```
EMPLOYE(M) = EMPLOYE(N)
et
EMPLOYE(M).NOM = EMPLOYE(N).NOM
```

Figure 5.

Pour faire cela en Forth, nous créons un tableau d'éléments et utilisons la forme d'allocation dynamique de la déclaration de structure comme dans la figure 6.

```
1800 18 S.ARRAY EMPLOYE
    18 DS
      S.MOVE EMPLOYE.MOVE
    16 S.FLD NOM
      FLD.MC NOM.MOVE
     2 S.FLD AGE
```

Figure 6.

```
N à EMPLOYE M à EMPLOYE
  EMPLOYE.MOVE
et
N à EMPLOYE M à EMPLOYE NAME.MOVE
```

Figure 7.

La figure 7 est donc la version Forth de la figure 5.

La seconde généralisation est d'avoir plus d'un niveau, comme dans la figure huit. Maintenant une référence à un champ individuel ressemble à :
EMPLOYE(N).NOM.INITIALES

```
DECLARE 1 EMPLOYE(100),
        2 NOM,
         3 INITIALES CHARACTER(2),
         3 PATRONYME  CHARACTER(14),
        2 AGE FIXED BINARY(15,0);
```

Figure 8.

En Forth, on peut faire la même chose en réinitialisant le décompte des adresses des champs, en déclarant la structure principale en premier, puis en déclarant les champs, comme dans le figure 9.

```
1800 18 S.ARRAY EMPLOYE
    18 DS
      S.MOVE EMPLOYE.MOVE
    16 S.FLD  NOM
      FLD.MC NOM.MOVE
     2 S.FLD AGE
 0    NOM SF
     2 S.FLD INITIALES
    14 S.FLD PATRONYME
```

Figure 9.

Alors en Forth, la référence aux champs se fait par:
N à EMPLOYE INITIALES

Une application typique d'une structure de données est une table avec deux champs : KY le champ clé, et VAL le champ valeur. Le tableau doit être maintenu trié selon le champ KY quand de nouvelles entrées sont ajoutées. Cela signifie que lorsque l'on ajoute une nouvelle entrée, une partie du travail concernera seulement le champ clé (c.a.d. le test pour trouver où la nouvelle entrée doit aller), tandis que l'autre partie concernera les deux champs (c.a.d. déplacer les autres entrées).

Au tableau de la figure dix est associé un compteur ACT qui donne le nombre actuel d'entrées utilisées. Le corps du tableau a deux champs, KY et VAL, tous les deux de 16 octets. £E est le nombre d'entrées maximum et a été défini comme une constante sur des bases générales de bonne programmation. On fait la convention que le tableau contient constamment une entrée bidon avec la plus grande valeur de VAL possible. Cela simplifie la recherche, puisque quand on ajoute une nouvelle entrée, on peut être sur qu'on arrivera tôt ou tard sur un champ clé plus grand.

TBL.INIT initialise la table, et insère l'entrée bidon en première position, pour que la condition supposée par la routine de recherche soit vérifiée. TBL.F prend une entrée à ajouter sur la pile, et compare son champ clé avec les champs clés des entrées du tableau, jusqu'à ce qu'une entrée avec un champ clé plus grand soit trouvé. TBL.F laisse alors l'indice sur la pile. TBL.I prend un indice sur la pile et décale toutes les entrées suivante d'un cran. Enfin, le mot TBL.INSERT supervise toute l'opération. Il vérifie si il reste de la place dans le tableau, et retourne 1 si ce n'est pas le cas. Sinon, il utilise TBL.F pour voir où l'entrée doit aller. Il utilise alors l'indice deux fois, une comme argument de TBL.I pour dégager de la place, et une autre comme argument de TBL.MOVE pour stocker la nouvelle valeur à l'endroit voulu. Finalement, il retourne 0 pour indiquer que l'entrée a bien été ajoutée au tableau.

```
1 VARIABLE ACT
25 CONSTANT £E
£E 4 * 4 S.ARRAY TBL
4 DS S.MOVE TBL.MOVE
2 S.FLD KY  2 S.FLD VAL

: TBL.F ACT à 0 DO
   DUP KY à I TBL KY à <
   IF DROP I LEAVE THEN LOOP ;

: TBL.I 1 - £E 1 - DO
   I TBL I 1 + TBL TBL.MOVE -1 +LOOP ;

: TBL.INSERT ACT à £E = IF DROP 1 ELSE
   DUP TBL.F DUP TBL.I TBL
   TBL.MOVE 0 THEN ;

: TBL.INIT 32767 0 TBLK KY ! 1 ACT ! ;
```

Figure 10.

Le fonctionnement des définitions.

Comme pour tous les mots CREATE ... DOES>, il y a une action au moment de la définition et une action au moment de l'exécution. Dans ce cas, l'action au moment de la définition comprend une sauvegarde des informations pour d'autres mots qui pourraient être utilisés plus tard, et un stockage dans le dictionnaire. Tous les mots utilisent trois variables pour sauvegarder des informations au moment de la définition:
- TLEN longueur totale de la structure
- CLEN longueur de la structure, jusqu'au champ sur lequel on travaille
- FLEN longueur du champ sur lequel on travaille

Ces variables sont utilisées pour calculer les données sauvées par les différentes routines de transfert (MOVE), donnée qui est utilisée par la partie (commune) DOES> des routines. L'exemple le plus simple est S.MOVE, qui transfère une structure entière. Quand une opération comme S.MOVE EMPLOYE.MOVE est exécutée, le compilateur Forth utilise TLEN à , pour mettre la valeur courante de TLEN dans EMPLOYE.MOVE. Quand

EMPLOYE.MOVE est exécuté, l'adresse de EMPLOYE.MOVE est empilée et la partie DOES> de S.MOVE est exécutée, et provoque un CMOVE qui transfère la structure entière.

Il y a juste quelques remarques à faire au sujet de l'utilisation des mots individuels. Il faut utiliser SS pour allouer une structure statique et commencer la définition des structures, seulement quand vous voulez à la fois les fonctions d'allocation et la structure de données. Quand la structure est déjà allouée, par un moyen ou un autre, il suffit d'utiliser DS pour commencer une structure dynamique. (La structure peut être déjà créée par exemple si elle a été définie par S.ARRAY, ou si on a créé un sous-champ, c'est à dire si on a relancé le décompte des adresses relatives).

Il peut sembler inutile d'avoir quatre routines pour transférer des champs (entre deux structures similaires, d'un champ à une adresse arbitraire, d'une adresse arbitraire vers un champ ou entre deux adresses arbitraires), mais on a besoin de toutes ces fonctions à un endroit ou à un autre; Je n'ai pas trouvé de noms plus astucieux pour les quatres cas. Les mots auraient pu être plus simples en passant des paramètres[7], mais cela aurait ralenti l'exécution, et on risque d'appeler beaucoup les mots définissant les champs.

Description détaillée des mots.

Puisque la plupart des mots ont à la fois une action à la compilation et une action à l'exécution, chaque description a jusqu'à trois parties : Premièrement, l'entrée créée par ce mot dans le dictionnaire; deuxièmement l'action à la compilation; troisièmement l'action à l'exécution. Pour les mots plus complexes, il y a une description pas-à-pas du mot, avec des descriptions de la pile entre parenthèses.

INIT.SV (run time) Sauve la longueur totale de la structure dans TLEN, initialise le pointeur au début du champ courant, et stocke 0 dans CLEN.

STR.SV (run time) Sauve l'offset du champ et sa longueur; appelé par les mots de compilation et crée une entrée dictionnaire.
(entrée dictionnaire)
mot 1 = offset du champ
mot 2 = longueur du champ

SS (run time) alloue un tableau, puis appelle INIT.SV

DS (run time) synonyme de INIT.SV

S.ARRAY (compile time) Sauve la taille d'un élément, puis alloue un tableau:
(entrée dictionnaire)
mot 1 = taille d'un élément
mot 2 = nb d'éléments
(run time) multiplie la taille d'un élément par l'indice, et ajoute la base.

S.MOVE (compile time) Sauve la longueur totale :
(entrée dictionnaire)
mot 1 = longueur totale
(run time) Récupère la longueur qui avait été sauvée et utilise CMOVE (les adresses de départ et d'arrivée doivent déjà être sur la pile).

S.FLD (compile time) Sauve l'offset de début de ce champ, stocke la longueur du champ dans FLEN et l'ajoute au pointeur courant :
(entrée dictionnaire)
mot 1 = offset du champ.
(run time) Lit l'adresse relative du champ, et l'ajoute à l'adresse de base.

FLD.MA (compile time) Appelle STR.SV pour sauver les caractéristiques du champ :
(entrée dictionnaire)
mot 1 = offset du champ
mot 2 = longueur du champ
(run time) (adresse - adresse de la structure - pointeur dictionnaire)
DUP : Duplique l'adresse de l'entrée dans le dictionnaire.
2+ : Pointe vers la longueur du champ.
>R : Sauve l'adresse de la longueur du champ sur la pile de retour.
à : Lit l'offset du champ.
+ : Ajoute l'offset du champ à l'adresse de début de la structure.
R> : Récupère l'adresse de la longueur du champ.
à : Récupère la longueur du champ.
(adresse absolue du champ - longueur du champ)
CMOVE : transfère depuis l'adresse fournie sur la pile vers la structure de données.

FLD.MB (compile time) Appelle STR.SV pour sauver les caractéristiques du champ :
(entrée dictionnaire)
mot 1 = offset du champ
mot 2 = longueur du champ
(run time) (adresse de la structure - adresse - pointeur dictionnaire)
>R : Sauve le pointeur vers le dictionnaire.
SWAP : Echange les adresses des structures.
R : Récupère le pointeur vers le dictionnaire.
à : Lit l'offset du champ.
+ : Ajoute l'offset du champ à l'adresse de début de la structure.
R> : Récupère l'adresse de la longueur du champ.
à : Récupère la longueur du champ.
(adresse - adresse absolue du champ - longueur du champ) CMOVE : transfère depuis l'adresse fournie sur la pile vers la structure de données.

FLD.MC (compile time) Appelle STR.SV pour sauver les caractéristiques du champ :
(entrée dictionnaire)
mot 1 = offset du champ
mot 2 = longueur du champ
(run time) (adresse de la structure 1 - adresse de la structure 2 - pointeur dictionnaire)
>R : Sauve le pointeur vers le dictionnaire.
R : Récupère le pointeur vers le dictionnaire.
à : Lit l'offset du champ.
+ : Ajoute l'offset du champ à l'adresse de début de la structure 2.
(adresse de la structure 1 - adresse absolue du champ de la structure 2)
SWAP : Echange les adresses.
R : Récupère le pointeur vers le dictionnaire.
à : Lit l'offset du champ.
+ : Ajoute l'offset du champ à l'adresse de la structure 1.
SWAP : echange les adresses
R> : Prend le pointeur vers le dictionnaire.
2+ : Pointe vers la longueur du champ.
à : Lit la longueur du champ.

suite page 14

```
1
\ Mots:  CLS HOME CASE OF ENDOF ENDCASE                09dec86 MP
: CLS  12 EMIT  #LINE OFF  #OUT OFF ;
 ' CLS IS DARK
: HOME 30 EMIT  #LINE OFF  #OUT OFF ;

: CASE CSP @ !CSP ; IMMEDIATE

: OF   COMPILE OVER  COMPILE =  COMPILE ?BRANCH  >MARK
COMPILE DROP ; IMMEDIATE
: ENDOF COMPILE BRANCH  >MARK  SWAP
>RESOLVE ; IMMEDIATE
: ENDCASE COMPILE DROP
BEGIN    SP@ CSP @ <>
WHILE    >RESOLVE
REPEAT
CSP ! ; IMMEDIATE
```

```
11
Commentaires de CLS HOME CASE OF ENDOF ENDCASE        09dec86 MP
CLS     ( ---)  efface le contenu de l'affichage video
        sur AMSTRAD/SCHNEIDER CPC $464, 664 et 6128.
HOME    ( ---)  ramene le curseur en ligne 0, colonne 0
        sur AMSTRAD/SCHNEIDER CPC $464, 664 et 6128.

CASE OF ENDOF ENDCASE sont des mots gerant une structure de con-
        trole, dont voici un exemple:

 : jour ( n ---)
case 1 of ." Lundi "     endof
     2 of ." Mardi "     endof
     3 of ." Mercredi"   endof
    ...etc...
     7 of ." Dimanche "  endof
endcase ;                        et '4 jour' affiche 'Jeudi'
```

```
2
\ Mots: PLOT PLOTR DRAW DRAWR
HEX
CODE PLOT
D1 C, E1 C, C5 C, CD C, BE9B ,
BBEA , C1 C, NEXT END-CODE
CODE PLOTR
D1 C, E1 C, C5 C, CD C, BE9B ,
BBED , C1 C, NEXT END-CODE
CODE DRAW
D1 C, E1 C, C5 C, CD C, BE9B ,
BBF6 , C1 C, NEXT END-CODE
CODE DRAWR
D1 C, E1 C, C5 C, CD C, BE9B ,
BBF9 , C1 C, NEXT END-CODE
DECIMAL
```

```
12
Commentaires de PLOT PLOTR DRAW DRAWR                 09dec86 MP

PLOT   ( x y ---) Affiche un point graphique aux coordonnes x y
       dans la couleur selectionnee par GPEN.

PLOTR  ( dx dy ---) Affiche un point graphique aux coordonnees
       dx dy relatives au trace du precedent point

DRAW   ( x y ---) Trace un trait graphique depuis le dernier
       point affiche vers le point de coordonnees x y.

DRAWR  ( dx dy ---) Trace un trait graphique depuis le dernier
       point affiche vers le point de coordonees relatives dx dy
```

```
3
\ Mots: TAG TRANS #WINDOW WINDOW                       09dec86 MP
HEX
CODE TAG
E1 C, C5 C, 7D C, CD C, BE9B ,
BB63 , C1 C, NEXT END-CODE
: TRANS
 22 EMIT EMIT ;
CODE #WINDOW
E1 C, C5 C, 7D C, CD C, BE9B ,
BBB4 , C1 C, NEXT END-CODE
DECIMAL
: WINDOW
26 EMIT  1- EMIT  1- EMIT  1- EMIT  1- EMIT ;
```

```
13
Commentaires de TAG TRANS #WINDOW WINDOW                    09
TAG     ( u ---) Selectionne mode ecriture texte en coordonnees
        graphiques: u=1 selectionne,  u=0 deselectionne

TRANS   ( u ---) Selectionne mode trace transparent graphique et
        texte:  u=1 selectionne,  u=0 deselectionne
#WINDOW ( u ---) Selectionne le numero de fenetre courant
        u compris dans l'intervalle 0..7


WINDOW  ( u1 u2 u3 u4 ---) Definit la taille de la fenetre cou-
        rante. Exemple:

  3 #WINDOW  25 20 60 30 WINDOW
```

```
4
\ Mots: MODE PAPER PEN INVERSE INK BORDER LOCATE       09dec86 MP
: MODE
 4 EMIT EMIT ;
: PAPER
 14 EMIT EMIT ;
: PEN
 15 EMIT EMIT ;
: INVERSE
 24 EMIT ;
: INK
 28 EMIT EMIT EMIT EMIT ;
: BORDER
 29 EMIT EMIT EMIT ;
: LOCATE
 31 EMIT EMIT EMIT ;
\ ' LOCATE IS AT   ( En option)
```

```
14
Commentaires de MODE PAPER PEN INVERSE INK BORDER LOCA09dec86 MP
MODE    ( u ---) Selection du mode d'affichage:
        u=0 20 car/  u=1 40 car/  u=2 80 car/ligne
PAPER   ( u ---) selection couleur fond en mode texte
        u compris dans l'intervalle 0..26
PEN     ( u ---) selection couleur forme en mode texte
        u compris dans l'intervalle 0..26
INVERSE ( ---) Inversion des couleurs fond et encre en mode
        texte.
INK     ( u1 u2 u3 ---) Etablit couleurs u1 u2 de l'encre u3:
        u1 u2 compris 0..26, u3 compris 0..15
BORDER  ( u1 u2 u3 ---) Etablit les couleurs u1 u2 du bord de
```

FORTH 83-Standard sous CP/M pour AMSTRAD CPC 464/664/6128

5

```
\ Mots: FLASH ?CHAR CLG ORIGIN                         09dec86 MP
HEX
CODE FLASH
E1 C, D1 C, C5 C, 63 C, CD C, BE98 ,
BC3E , C1 C, NEXT END-CODE
CODE ?CHAR
C5 C, CD C, BE98 ,
BB60 , 26 C, 00 C, 6F C, C1 C, E5 C, NEXT END-CODE
CODE CLG
C5 C, CD C, BE98 ,
BBDB , C1 C, NEXT END-CODE
CODE ORIGIN
D1 C, E1 C, C5 C, CD C, BE98 ,
BBC9 , C1 C, NEXT END-CODE
DECIMAL
```

15

```
\ Mots: FLASH ?CHAR CLG ORIGIN                         09dec86 MP

FLASH  ( u1 u2 ---) Etablit la duree des clignotements. u1 et u2
       sont donnes en 1/50e de seconde.

?CHAR  ( --- c) Delivre le code du caractere situe a l'emplace-
       ment du curseur. Exemple: 10 15 LOCATE ?CHAR

CLG    ( ---) Vide le contenu de la fenetre graphique.

ORIGIN ( y x ---) Precise l'origine de la fenetre graphique.
       Soit 0<=y<=399 et 0<=x<=639, exemple:
       399 200 639 300 GWINDOW  200 300 ORIGIN
```

6

```
\ Mots: GPEN GPAPER GWINDOW VSWAP                      09dec86 MP
HEX
CODE GPEN
E1 C, C5 C, 7D C, CD C, BE98 ,
BBDE , C1 C, NEXT END-CODE
CODE GPAPER
E1 C, C5 C, 7D C, CD C, BE98 ,
BBE4 , C1 C, NEXT END-CODE
CODE GWINDOW
E1 C, D1 C, C5 C, CD C, BE98 ,
BBCF , C1 C, E1 C, D1 C, C5 C, CD C, BE98 ,
BBD2 , C1 C, C3 NEXT END-CODE
CODE VSWAP
E1 C, D1 C, C5 C, 45 C, 48 C, CD C, BE98 ,
BBB7 , C1 C, NEXT END-CODE
DECIMAL
```

16

```
\ Mots: GPEN GPAPER GWINDOW VSWAP                      09dec86 MP

GPEN   ( u ---) Etablit la couleur u du curseur graphique.
       Soit 0<=u<=25<

GPAPER ( u ---) Etablit la couleur de fond en mode graphique.
       Soit 0<=u<=25.

GWINDOW ( u1 u2 u3 u4 ---) Determine les coordonnees de la fe-
        netre graphique. u1=y haut, u2=y bas,
                         u3=x droite, u4=u gauche.

VSWAP  ( u1 u2 --- ) Echange les caracteristiques des fenetres
       u1 et u2.
```

7

```
\ Mots: MOVE MOVER
HEX
CODE MOVE
D1 C, E1 C, C5 C, CD C, BE98 ,
BBC0 , C1 C, NEXT END-CODE
CODE MOVER
D1 C, E1 C, C5 C, CD C, BE98 ,
BBC3 , C1 C, NEXT END-CODE
DECIMAL
```

17

```
\ Mots: MOVE MOVER

MOVE   ( y x ---) Deplace le curseur aux coordonnees y x.

MOVER  ( y x ---) Deplace le curseur aux coordonnees
       relatives y x.
```

8

```
\ Table des sinus 16 bits
HERE
0000 , 0175 , 0349 , 0523 , 0698 , 0872 , 1045 , 1219 ,
1392 , 1564 , 1736 , 1908 , 2079 , 2250 , 2419 , 2588 ,
2756 , 2924 , 3090 , 3256 , 3420 , 3584 , 3746 , 3907 ,
4067 , 4226 , 4384 , 4540 , 4695 , 4848 , 5000 , 5150 ,
5299 , 5446 , 5592 , 5736 , 5878 , 6018 , 6157 , 6293 ,
6428 , 6561 , 6691 , 6820 , 6947 , 7071 , 7193 , 7314 ,
7431 , 7547 , 7660 , 7771 , 7880 , 7986 , 8090 , 8192 ,
8290 , 8387 , 8480 , 8572 , 8600 , 8746 , 8829 , 8910 ,
8988 , 9063 , 9135 , 9205 , 9272 , 9336 , 9397 , 9455 ,
9511 , 9563 , 9613 , 9659 , 9703 , 9744 , 9781 , 9816 ,
9848 , 9877 , 9903 , 9925 , 9945 , 9962 , 9976 , 9986 ,
9994 , 9998 , 10000 ,
: SINNEG DUP 90 > IF  180 SWAP -  THEN [ 0 SWAP ] LITERAL
SWAP 2 * + @ ; DROP
```

18

```
\ Table des sinus 16 bits

Cette table a ete composee a partir de la formule:

 n=int(10000*sin(a))
 pour 0<=a<=90

SINNEG  Delivre le sinus signe.
```

```
    9
\ Fonctions trigonometriques
: SIN
 360 MOD DUP 0< IF 360 + THEN
DUP 180 > IF 180 - SINNEG -1 * ELSE SINNEG THEN ;
: *SIN
 SIN 10000 */MOD SWAP 5000 > IF 1 + THEN ;
: *COS
 90 + *SIN ;
VARIABLE X0   VARIABLE Y0   VARIABLE DEB  VARIABLE FIN
: CENTER
 Y0 !  X0 ! ;
: POSX1Y1
 OVER OVER *COS  X0 @ +
 ROT ROT *SIN Y0 @ + ;
```

```
   19
\ Fonctions trigonometriques
SIN     delivre le sin(a modulo 360)


*SIN    ( n deg --- sin) Delivre le sinus d'un angle multiplie
        par n. Exemple:  100 60 *SIN .  affiche 50
*COS    idem a *SIN mais applique au cosinus.

X0 Y0 DEB FIN  variables utilisees pour le trace d'arc de cercle
CENTER  ( x y ---) Affecte le centre d'un arc de cercle.

POSX1Y1  Calcule la position d'un point par rapport au centre
         lors d'un trace d'arc de cercle.
```

suite de la page 11

CMOVE : Transfert.
FLD.MD (compile time) Sauve la longueur du champ.
(entrée dictionnaire)
mot 1 = longueur du champ
(run time) Lit la longueur du champ et effectue le transfert entre les deux adresses.

Bibliographie

1. Beers, David A. "Quadruple Word Simple Arithmetic" *Forth Dimensions IV/1.*
2. Clark, Alfred, "Complex Analysis in Forth" *Forth Dimensions III/4.*
3. Harris, Kim, "Forth Extensibility" *BYTE, Vol 5 No 8 (Aout 1980).*
4. Haydon, Glen B.,"Elements of Forth Data Base Design", *Forth Dimensions III/2.*
5. Helmers, Peter H. "Userstack" *Forth Dimensions III/1.*
6. McCourt, Michael and Marisa, Richard A., "The String Stack", *Forth Dimensions III/4.*
7. McKibbin, David, "Parameter Passing to DOES>", *Forth Dimensions III/1.*

Impression graphique multi-écrans par M. ANZIL

```
ECRAN No     1 FENETRE
FORGET ME   : ME ; DECIMAL
  0 VARIABLE Curs 0 , 4 CONSTANT 4
 10 CONSTANT D_OC     20 CONSTANT N_OC
 50 CONSTANT D_Y     100 CONSTANT N_Y
 : Cadre   2OVER D+ OVER 4 PICK 2OVER
LINE 2OVER TO 4 ROLL OVER TO TO DROP ;
 : CURS! 65217 VC@ 65218 VC@ curs 2! ;
 ( sauvegarde position du curseur)
 : CURS@ curs 2@ SWAP AT ;    CURS!
 : >PIm D_OC D_Y     N_OC N_Y  BPUSH ;
 : ↑win D_OC 4 * 4 + D_Y N_Y + 4 -
        N_OC 2- 4 *  N_Y 8 - WINDOW ;
 : dims D_OC 4 * 2+  D_Y 2+
        N_OC 4 * 4 - N_Y 4 - ;
 : >cst     ' N_Y ! ' N_OC !
            ' D_Y ! ' D_OC ! ;
 : Fen STANDARD 0 BRIGHT
       1 EMIT >cst >PIm ↑win
       2 INK dims  2OVER D+ BOX
       3 INK dims cadre
                 PAPER PAGE PEN ; -->
ECRAN No     2 Echange d'écrans
 : Sortie     BPOP 0 PAPER 2 PEN
          1 BRIGHT 1 EMIT CURS@ ;
 : 1ECR   3 1   20  20 32  40 Fen ;
 : 2ECR   3 2   20 100 32  80 Fen ;
 : 3ECR   2 1   32  50 26 100 Fen ;
 ( nom  Pen  PaPer  octet/X/origine
   Pixel/Y/origine  octet/X/long
   Pixel/Y/haut -- Fen )

      0 VARIABLE STOCK 7296 ALLOT
   7296 CONSTANT 1/2e  ( 1/2 écran    )
  49152 CONSTANT Vidéo ( départ vidéo)
  34048 CONSTANT Ch/II ( CHROMA II    )
 : -><-             ( échange de page )
 Video STOCK 1/2e VCMOVE
 Ch/II video 1/2e VCMOVE
 STOCK Ch/II 1/2e VCMOVE
 Video 1/2e + STOCK 1/2e VCMOVE
 Ch/II 1/2e + Video 1/2e + 1/2e VCMOVE
 STOCK Ch/II 1/2e + 1/2e VCMOVE
 Video STOCK 1/2e VCMOVE ;            -->
ECRAN No     3 VOIR
 : >SAUV
   Video    1/2e + STOCK   1/2e VCMOVE
   STOCK
  [ Ch/II 1/2e + 1/2e + 64 + ] LITERAL
                  1/2e  CMOVE
   Video   STOCK    1/2e VCMOVE ;
 : SAUV>
   STOCK Video 1/2e VCMOVE
  [ Ch/II 1/2e + 1/2e + 64 + ] LITERAL
   STOCK 1/2e CMOVE
   STOCK [ Video 1/2e + ] LITERAL
                  1/2e VCMOVE ;
 : R/IN  sortie >SAUV CLS 1ECR ;
 : R/OUT sortie CLS SAUV> 1ECR ;
( R/IN sauve l'écran dans STOCK et en
  fin de mémoire , R/OUT l'affiche )
```

```
 : VOIR BPOP -><- 1ECR ." ■■ VOIR"
       CR ." ■■ ++++" QUIT ;
     BINIT 1ECR ." ■■ VOIR"
           CR ." ■■ ----" CR          -->
ECRAN No     4 DBLCOP
 0 VARIABLE #A
 : ↑↑  I' 0= IF 1 ELSE I' 1 =
              IF 2 ELSE I' 0 DO 2 LOOP
   I' 1- 0 DO * LOOP THEN THEN ;

 : DBLCOP FAST sortie
          CR CR 0 #A ! 8 LEMIT
1 227 DO I
  241 1 DO 7 0 DO J OVER I - POINT 0=
             IF ↑↑ #A +! THEN LOOP
             #A à 128 + LEMIT 0 #A !
        LOOP -><-
  241 1 DO 7 0 DO J OVER I - POINT 0=
             IF ↑↑ #A +! THEN LOOP
             #A à 128 + LEMIT 0 #A !
        LOOP -><-
           DROP 13 LEMIT
 GET 13 = IF LEAVE THEN
-7 +LOOP
    15 LEMIT 13 LEMIT 1ECR SLOW ;   -->

ECRAN No     5 ExPlications
( HECTOR HRX ____________ Jean Michel
  ________________________   ANZIL
  Fenêtrage à la demande avec sauve-
garde de la position du curseur et
sauvegarde de la portion d'écran dans
la pile image
             "sortie" = retour écran .
 DBLCOP permet d'imprimer deux écrans
 sur toute la largeur de l'imprimante
 SEIKOSHA GP 500A en mode graphique
 480 points . Cette version est adap-
 tée au logiciel graphique CHROMA II
 écrit en BASIC adresse dessin:34048
 Fonctionnement: 1er écran chargé par
 RESET/INIT et l'option 2 du menu HRX
 "K7" . Tapez VOIR et sauvez par R/IN
 Chargez le 2eme par INIT et option 2
 Tapez R/OUT pour revoir le 1er
 tapez VOIR pour vérifier que tout le
 monde est là puis DBLCOP pour impri-
 mer les deux pages .       Nov*1986)
```

HECTOR HRX

Exemple d'exécution de DBLCOP

PORTEFEUILLE A ABATTANTS	#A/N1	COLLEGE	CLAS.
NOM. .... PRENOM			
OPERATIONS	CROQUIS	OBSERVATIONS	OUTILS
10 DEBIT CARTON 2MM R1 R8 CARTON 1MM R7 R8 R13 R14 TOILE ⇒ COUVRURE R6 460 × 340 mm	60 285 310 R16 R R11 R10 R17 R16 3 R5 R4 60 70 225 130 285 320 TOILE 1030 X 130mm	ATTENTION R4=R5= 70 MM R10=R11=R16= 50 MM	
20 TRACAGE TRACER SUR R8 4 RESERVES DE 20mm 4 TRIANGLES RECT. ISOC. DE 80mm DE COTE	R 8 12 45° R3 10		

Opérations en notation scientifique

par G. SOULA

```
( DIVISION NOTATION SCIENTIFIQUE )

2VARIABLE N1    VARIABLE N2    VARIABLE DIV    VARIABLE EXP
VARIABLE P1     VARIABLE P2    VARIABLE QUOT 10 ALLOT

: F1    2INPUT  N1 2! ;
: F2    INPUT DUP 32766 U<
   IF N2 !
   ELSE . ." TROP GRAND! " THEN ;

: QUOTIENTS
   DUP DIV !  5 0
   DO U/ I 2* QUOT + !
      10 U* DIV @
   LOOP  DROP 2DROP ;

```

```
( DIVISION NOTATION SCIENTIFIQUE )

: 4DECIM    5 0
   DO QUOT I 2* + @
      I DUP 0=
      IF DROP 10000 U*
      ELSE DUP 1 =
            IF DROP 1000 U*
            ELSE DUP 2 =
                  IF DROP 100 U*
                  ELSE DUP 3 =
                        IF DROP 10 U*
                        ELSE DUP 4 =
                              IF DROP 1 U* THEN
      THEN THEN THEN THEN
   LOOP D+ D+ D+ D+ ;
```

```
( DIVISION NOTATION SCIENTIFIQUE )
: E1   ." E? " INPUT P1 ! ;        : E2    ." E? " INPUT P2 ! ;
: 1ER-NB   ." 1ERNB? "  F1 E1 ;
: 2EME-NB  ." 2EME NB? "  F2 E2 ;
: ENTREE   CR 1ER-NB  CR 2EME-NB ;
: CALCULS
  ENTREE N1 2@ N2 @ QUOTIENTS
  4DECIM  P1 @ P2 @ - EXP ! ;
: AFFICHE
  CR <# 69 HOLD # # # # 46 HOLD #S #> TYPE EXP @ . ;
: F/   CALCULS AFFICHE ;
```

```
( MULTIPLICATION NOTATION SCIENTIFIQUE )
VARIABLE COMPTER                     VARIABLE EXP
VARIABLE N1 20 ALLOT                 VARIABLE N2 20 ALLOT
: F1
  ." 1ER NB? "  N1 12 EXPECT  0 EXP !
  12 0 DO N1 I + C@ 0= IF LEAVE I COMPTER ! THEN LOOP
  N1 1- NUMBER
  COMPTER @ ;
: F2
  ." 2EME NB? " N2 12 EXPECT
  12 0 DO N2 I + C@ 0= IF LEAVE I COMPTER ! THEN LOOP
  N2 1- NUMBER
  COMPTER @ ;
```

```
( MULTIPLICATION NOTATION SCIENTIFIQUE )
: OPAD    PAD 20 ERASE ;
: REDUIT
  DUP 9 >
  IF ." NOMBRE TROP GRAND "
     DROP 2DROP QUIT
  ELSE DUP 4 > IF 4 - DUP EXP +!
                  0 DO # LOOP
               ELSE DROP 0 EXP +!
  THEN         THEN
  OPAD ;
```

```
( MULTIPLICATION NOTATION SCIENTIFIQUE )
: F*
  CR F1 REDUIT
  CR F2 REDUIT
  DD* 10000 U/
  SWAP 5 > IF 1+ THEN 0 ;
: AFFICHE   <# 69 HOLD #S #> TYPE  EXP @ 4 + .  ;
: F*.   F* AFFICHE ;
```

```
( NOTATION SCIENTIFIQUE )
: ?DPL
  DPL @
  DUP 0<
  IF DROP 0
  ELSE DROP -1
  THEN ;
: CC
  ?DPL 0
  BEGIN 1+ DUP
        TIB @ + C@ DUP 32 =
        IF   DROP 1
        ELSE DROP 0 THEN
  UNTIL ;
```

```
( NOTATION SCIENTIFIQUE )
VARIABLE EXP
: AFFICHE
  <# 69 HOLD #S #>
  TYPE
  EXP ? ;

  1 SWAP ?DUP
  IF 0 DO OVER * LOOP
  THEN
  SWAP DROP ;
```

```
( NOTATION SCIENTIFIQUE )
: (F)
  0 EXP !
  CC DUP 9 >
  IF ." NOMBRE TROP ELEVE "
     DROP 2DROP
  ELSE DUP 5>
        IF 5 - DUP EXP !
           0 DO # LOOP AFFICHE
        ELSE DROP 0 EXP !
             AFFICHE
        THEN
  THEN ;
```

GESTION DE FICHIERS EN F83   par Charles CORBOU

La première chose à faire est de créer un fichier, en précisant sa taille en blocs:

```
CREATE-FILE <nom-de-fichiers>  ( n ---)
```
crée un fichier avec n blocs.

Dès lors, vous pourrez ouvrir le fichier:

```
OPEN <nom-de-fichier>
```

Mais OPEN fait bien mieux: il crée, si cela n'a pas été fait, le mot <nom-de-fichier> qui, à l'exécution, ouvrira le fichier. Exemple:

```
10 CREATE-FILE FICHIER1   crée FICHIER1
                          avec 10 blocs (0 à 9)
OPEN FICHIER1             ouvre FICHIER1 et
                          fabrique le mot FICHIER1
FICHIER1                  réouvre FICHIER1
```

Le mot MORE (n ---) ajoute n blocs au fichier existant. Attention, certaines versions "oublient" de sauvegarder la nouvelle configuration du fichier! Le remède est de redéfinir MORE comme suit:

```
: MORE   CAPACITY SWAP 8* MAXREC= +! CAPACITY
   DO I BLOCK B/BUF BLANK UPDATE LOOP FLUSH ;
```

La copie de blocs est exécutée à l'aide du mot COPY (b1 b2 ---), ce qui a pour action de copier le contenu du bloc b1 dans le bloc b2.

La copie de blocs multiples est exécutée par CONVEY et utilisé comme suit:

```
b1 b2 TO b3 CONVEY
```

Exemple:

```
10 20 TO 15 CONVEY
```

copie les blocs 10 à 20 dans les blocs 15 à 25. Selon le cas, CONVEY fera les copies dans l'ordre croissant ou décroissant, ceci pour éviter les chevauchements de blocs à copier. Vous n'avez donc pas à vous en préoccuper.

Le standard F83 prévoit la gestion simultanée de deux fichiers, le fichier courant et le fichier secondaire (IN-FILE) que nous appellerons fichier donneur. Les concepteurs du F83 ont prévu, dans le vocabulaire FILES, de nouvelles versions des mots LOAD COPY et CONVEY effectuant respectivement le chargement, la copie simple et la copie multiple à partir du fichier secondaire. Efficace, mais dangeureux pour les distraits oubliant quel est le fichier primaire et quel est le fichier secondaire!

Un mot permet d'ouvrir le fichier secondaire: FROM. Il s'utilise sous la forme:

```
FROM <nom-de-fichier>
```

Ce mot donne également accès au vocabulaire FILES et donc aux versions spéciales de LOAD COPY et CONVEY. Dans les versions 1xx du F83, ces mots ramènent au vocabulaire initial après exécution. Ils ne servent donc qu'une fois Dans les versions suivantes, ils restent actifs tant que vous n'vez pas explicitement quitté le vocabulaire FILES. Exemple, première version de F83:

```
FORTH
OPEN FICHIER1  ouvre FICHIER1
FROM FICHIER2  ouvre FICHIER2 et
               pointe sur le vocabulaire FILES
1 1 COPY       copie le bloc 1 de fichier 2
               dans le bloc 1 de FICHIER1 et ramè-
               ne au vocabulaire FORTH.
3 4 COPY       copie le bloc 3 de FICHIER1
               dans le bloc 4 de FICHIER1.
```

Exemple pour les versions suivantes:

```
FORTH
OPEN FICHIER1  ouvre FICHIER1
FROM FICHIER2  ouvre FICHIER2 et
               pointe sur le vocabulaire FILES
1 1 COPY       copie le bloc 1 de fichier 2
               dans le bloc 1 de FICHIER1
3 4 COPY       copie le bloc 3 de FICHIER2
               dans le bloc 4 de FICHIER1.
vous êtes toujours dans le vocabulaire FILES
```

Le mot SWITCH permute le fichier courant et le fichier donneur. Si vous n'en disposez pas, définissez-le comme suit:

```
: SWITCH
FILE @ IN-FILE @ FILE ! IN-FILE ! ;
```

Voici, pour votre tranquilité d'esprit, un mot qui ramènera les deux fichiers au seul fichier courant:

```
: UNIQUE FILE @ IN-FILE ! ;
```

Note: IN-FILE est une variable contenant le FCB du fichier donneur. Dans certaines versions, cette variable s'appelle >FROM. Remplacez donc, si besoin est, IN-FILE par >FROM dans les deux définitions ci-dessus. ■

MANIPULATIONS POUR SAUVEGARDE DES BLOCS SUR AMSTRAD PCW

par Marc PETREMANN

C'est bien joli, le FORTH 83-Standard est compatible en version CP/M sur toute la gamme AMSTRAD, mais c'était trop beau pour durer. Mr BRUN Nicolas a trouvé l'os; le mot DONE exécuté en fin d'édition ne sauvegarde rien.

C'est que Mr AMSTRAD, ne voulant rien faire comme tout le monde (il était très bien pourtant le CP/M 2.0) a imaginé d'embêter le peuple avec un CP/M 3.0 appelé aussi CP/M+. Et ce CP/M+ a comme principale propriété de ne pas permettre la compatiblité avec les logiciels tournant sous CP/M 2.0, cetains du moins...

En effet, le PCW8256 ou PCW8512 se sert de son extension mémoire comme d'un disque virtuel. Nous disposons donc sous CP/M+ de deux drives voir trois:

```
1 drive physique:     A:  M:
2 drives physiques:   A:  B:  M:
```

Or, quand vous éditez un programme sous FORTH, l'option W utilisée sous EDITOR ou SAVE-BUFFERS écrit le contenu de vos blocs sur M: et non sur A: ou B:. Donc, s'il vous prend l'envie de quitter FORTH puis d'y revenir, vous aurez la désagréable surprise de constater que rien n'a été sauvegardé. La solution à ce remède est simple, il suffit de taper:

```
DONE CP/M CLOSE FORTH
```

ou de définir:

```
ONLY CP/M ALSO FORTH ALSO
: TERMINE DONE CLOSE ;
ONLY FORTH ALSO
```

et d'utiliser TERMINE à la place de DONE. Cette définition n'est pas incompatible avec F83 sur AMSTRAD CPC464, CPC664 ou CPC6128. Sur IBM et compatibles, on remplacera CP/M par DOS. ■